What actually goes on inside an FFT?

Introduction

In the previous article in this series, we looked at digital filtering and showed how a buffer-full of input samples was manipulated to give a filtered version of the input waveform. Here, we look at another process that uses broadly similar 'hardware', but for a completely different purpose. This is the 'Fourier transform', used to generate frequency data from an input time waveform, and its high-speed computer-optimised variant, the 'fast Fourier transform' (FFT).

In the same way as is done for digital filtering, an input analogue waveform is sampled, using an analogue-to-digital converter (ADC) running at a sampling rate of F_s, into a time-ordered series of digital samples, and these are stored in a serial buffer. The buffer therefore contains the last N samples at any one time. For reasons that will be explained later, the buffer length is usually an exact power of two, like 512 or 65536, and is generally much longer than one used for digital filtering. Sometimes the buffer length can extend to over one million samples. The contents of the buffer are used in the FFT process. As in the case of a digital filter, each historical sample is multiplied by a coefficient and summed in a multiply-and-accumulate (MAC) process. But that is where the similarity ends.

The basic process

We want to extract frequency data from a time series of input samples. We do this by multiplying a block of samples of the unknown input by a series of sampled sine waves, one sine wave for each frequency component we wish to test for. To explain how this works, we need to look at the mathematics of multiplying two sine waves together. A sine wave of frequency F_1 can be expressed mathematically as $\sin(2\pi F_1 t)$, where t represents the time. If we have two sine waves, one of frequency F_1 and the other of frequency F_2, then the result, Y, of multiplying them can be written as

$$2Y = 2\sin(2\pi F_1 t)\sin(2\pi F_2 t),$$

and this can be re-expressed as the sum of two cosines

$$2Y = \cos(2\pi[F_1 + F_2]t) + \cos(2\pi[F_1 - F_2]t).$$

Now a cosine is the same as a sine but with a phase shift of 90°, so the output just contains two components with frequencies equal to $F_1 + F_2$ and $F_1 - F_2$. This is our standard frequency mixer process, with the well-known sum and difference terms shown mathematically.

Consider what happens if F_1 and F_2 are the same frequency, ie $F_1 = F_2$, (and to simplify things we will, for now, assume that the waves are in phase with each other). One output term is equal to $2F_1$, the second harmonic, and the other is equal to zero frequency, ie DC. In the FFT processing, the results are averaged over time, so the second-harmonic term disappears, since the average of a sine or cosine wave is zero. However, the DC term is a constant since $\cos(0) = 1$, and this does not average to zero. Thus, by multiplying the input waveform by a sine wave of frequency F_1 and then averaging the result, we pick out just the component in the input waveform which has a frequency of F_1.

By now it should be becoming apparent how the Fourier transform works. We multiply our input waveform by a sine wave of a given frequency, F, and average the result. We get a constant output if there is a component of the input waveform at frequency F, and zero output if there isn't. By stepping F over a range of frequencies, and repeatedly multiplying and averaging for each frequency, we map out the individual frequency components in the input waveform.

Figure 1 shows this graphically for the first five harmonic terms. The reference (shown in black at the top) is multiplied in turn with each of the light grey harmonics, to give the overlying coloured waveforms. It is clear that the average of the first one in red, where the test frequency is equal to the reference, averages to a positive value. The other traces, 2F, 3F etc, can be seen to average to zero.

Practical realities

In practice, we can't just 'multiply the input by a bunch of different reference sine waves'. We instead have to use a finite length block of input samples, and multiply these with a block of reference sine-wave samples. The block, or buffer, length can be anything we like but, because it is finite, it restricts the quality and accuracy of our results.

The lowest, or base, frequency we can derive from a block, or buffer-full of samples, is the case where the buffer contains exactly one complete cycle of samples of the sine wave of the reference. The next term is where the buffer contains two complete cycles, then three complete cycles then four and so on, up to the Nyquist limit where the reference is just a series of alternating plus and minus terms in each buffer location, equal to the base frequency times half the buffer length. This base frequency, equivalent to one cycle over the buffer length, is the resolution with which we can measure the frequencies in the calculated spectrum. Each term in the frequency data represents the amplitude of each multiple of this frequency up to half the sampling rate.

And this is the basis of the Fourier transform process; we fill a buffer of length N with a series of input samples, and repeatedly multiply each stored input sample with a series of reference sampled sine waves, starting at 1 and going to the N/2 harmonic, accumulating the results for each set of multiplications of each frequency component. The resulting single accumulated sum for each frequency gives the amplitude of the frequency component at that frequency. So the series of N/2 MAC results now corresponds to the amplitudes of each component of our input waveform. We have generated the spectrum of our input data, just what we wanted to do, up to a limit of half the buffer length multiplied by the base frequency. As this is being done on a discrete set of samples, it is usually referred to as the discrete Fourier transform, or DFT. Each frequency buffer location is termed a 'bin', as in dustbin, as each one contains a separate item of information, ie the amplitude of each frequency component. The width of each bin, or the 'bin size', is F_s / N, and this is the resolution to which we can measure each frequency component.

Rounding and in-between frequencies, and phase

What happens if a component of the input is not an exact multiple of the buffer length, but falls somewhere between one of the two reference frequencies, say at 3.5 times the base frequency? In that case, both the third and fourth reference terms will provide a non-zero averaged result. This will not be as much as if the exact frequency were to be hit, but some result will appear for both 3rd and 4th reference terms. In this case, other higher reference frequencies may also give some MAC output, but this will progressively reduce in amplitude as the harmonic number goes up. The situation can be ameliorated by 'windowing' the input data as we'll see later.

So far, we have assumed that the input frequency component was exactly in phase with the reference, so that $\cos(2\pi[F1 - F2]$

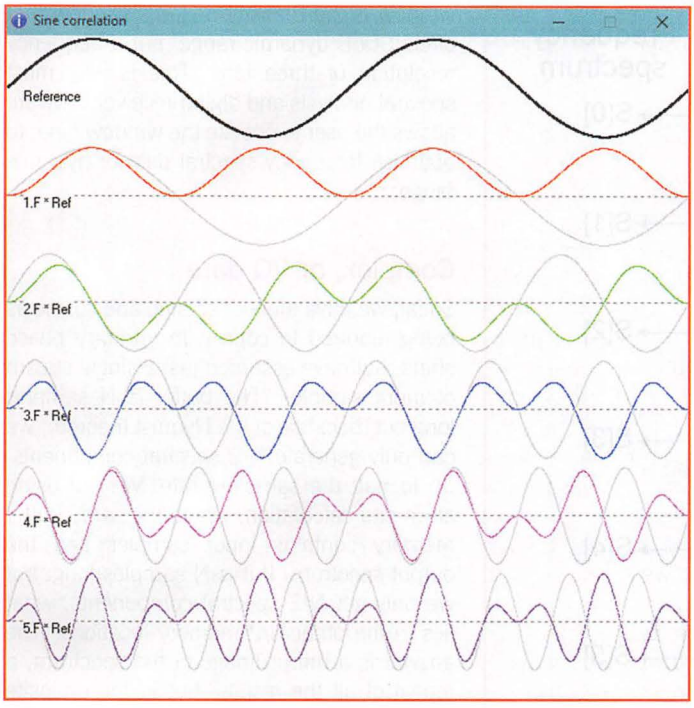

FIGURE 1: The correlation of sine waves. The top trace in black is multiplied by 1, 2, 3, 4 and 5 times its frequency, each shown in grey, then averaged over the entire width. Note how only the plot in red, 1F, gives a finite average value. All the others average to zero.

t)) = 1 for F1 = F2, but this will hardly ever be the case. In practice, we use both cosine and sine terms of the reference frequencies, and calculate a pair of results with a 90° phase difference between them. Let's assume our input is 90° out of phase with the original reference, ie $\cos(2\pi Ft)$. Now $2\cos(2\pi Ft)\sin(2\pi Ft) = 0$. But by simultaneously doing two MAC processes on each input sample, with cos and sin terms, or 'real' and 'imaginary' in the parlance of complex algebra, or 'I' (for in-phase) and 'Q' (for quadrature-phase) as electronic engineers call them, we generate a pair of results, $\sin(2\pi F1t).\sin(2\pi F2t)$ and $\cos(2\pi F1t).\cos(2\pi F2t)$ (the latter also gives a constant averaged output when the frequencies are the same). The two MAC results, when added together, then give the amplitude of the input component, whatever its phase. As the results are 90° apart, vector addition is needed to add the two numbers using Pythagoras, so the amplitude is $\sqrt{(I2 + Q2)}$. There is also the added advantage that, by comparing the amplitudes of the cosine and sine terms, we can calculate the phase. This is of little value if all we want is a spectrum trace, or waterfall display of the input, but is vital when the Fourier transform is used in the decoding of multi-carrier waveforms, as used for most modern digital signalling like terrestrial TV and DAB broadcasting, and 4G/5G mobile phone signals.

This use of paired cos/sin multiplications is analogous to the I/Q channels we are familiar with in so many other aspects of DSP, and we'll see later that feeding I/Q data into the FFT process actually helps.

Putting in some numbers

So far, we have referred to a 'base frequency', or lowest spectral component, that can be detected, equivalent to one complete sinusoidal cycle over the buffer length. In reality what we are doing is feeding in a series of samples at a sampling rate F_s. When the buffer is filled, its complete length is then equivalent to a 'base frequency' of F_s/N. Thus, if we are sampling at 12kHz and have a 1024-sample buffer, the frequency components are evaluated at multiples of 12000/1024 = 11.72Hz. A longer buffer length of 2^{20}, or 1048576 cells, gives us a resolution of 0.0114Hz, but requires over a million complex MAC operations to get there, just for one frequency component, or 'bin'.

If we bear in mind that N separate MAC operations are needed for each frequency component, and we are evaluating N/2 frequency terms (both I and Q at each frequency), it becomes evident that we need $N^2/2$ MAC operations to evaluate the complete spectrum using the DFT on each block of data. For a buffer length of 1024, that is about half a million complex I and Q MAC operations for every set of results. At 11.72Hz resolution, a new spectrum will be generated every 85 milliseconds, needing a processor capable of doing 500000/.085 or about 6 billion MAC operations per second just for this relatively-low resolution answer; this is barely feasible even with the fastest processing hardware. Doing this for the million-sample buffer is totally unrealistic.

It looks as if the DFT process just described is totally impractical for real-time use, other than for low-resolution frequency analysis run on fast processing hardware. But fortunately there is a solution, and it was discovered long before the advent of digital electronics, around the time of the sea battle of Trafalgar, by Carl Friedrich Gauss (see Wikipedia).

The fast Fourier transform (FFT)

In 1965, Messrs. Cooley and Tukey popularised Gauss's method whereby the matrix of multiplications needed for the DFT could be greatly reduced. They made use of the fact that, for multiple sine values, many numbers repeat, so a multiplication at one point of the input waveform by one reference sine value is equal to another multiplication at a different point on another. Their algorithm optimised this process, doing just exactly the number of MAC commands needed for the complete DFT. This reduced the number of commands needed from $N^2/2$ to $N.\log_2(N)$, which means that, for the 1024-point example described above, the number of operations reduces from around 500000 to 10240, fifty times faster. For longer buffer lengths, the increase is even more dramatic. A buffer size of 1 million points, instead of needing something like the completely impractical 10^{12} MAC operations which would take several hours to compute on a GHz-clocked processor, is now reduced to 20 million operations, which it can easily manage in the time it takes for the buffer to refill with the next set of data.

The FFT process is quite complex to visualise, and will not be described in fine detail here. In essence, input samples are reordered and broken down into a tree structure of multiple small DFTs of just two samples at a time, whose results are added together. This 2x2 minimal DFT is often referred to as a 'butterfly' because of its appearance when shown graphically, and involves no actual multiplication as the coefficients, after breaking things down this far, are either 1 or -1. The number of actual multiplications for the whole FFT is now vastly reduced. Once the processing through the tree has finished, the samples have to be reordered, and this involves reversing the bits of the binary number representing each cell. Alternatively, the bit-reversal process can be done on the input samples before the tree of small MAC processes. The entire process is often done in-situ, using the same buffer memory as the stored samples. The buffer array enters the FFT process with time-domain data, and exits containing frequency-domain (or 'spectral') data. The need for bit-reversal, and the way the tree is broken down ultimately to a set of repeated 2x2 multiplications, is the reason that the buffer length has to be a power of two: 256, 1024, 65536 etc.

Filtrate
radioecm@googlemail.com

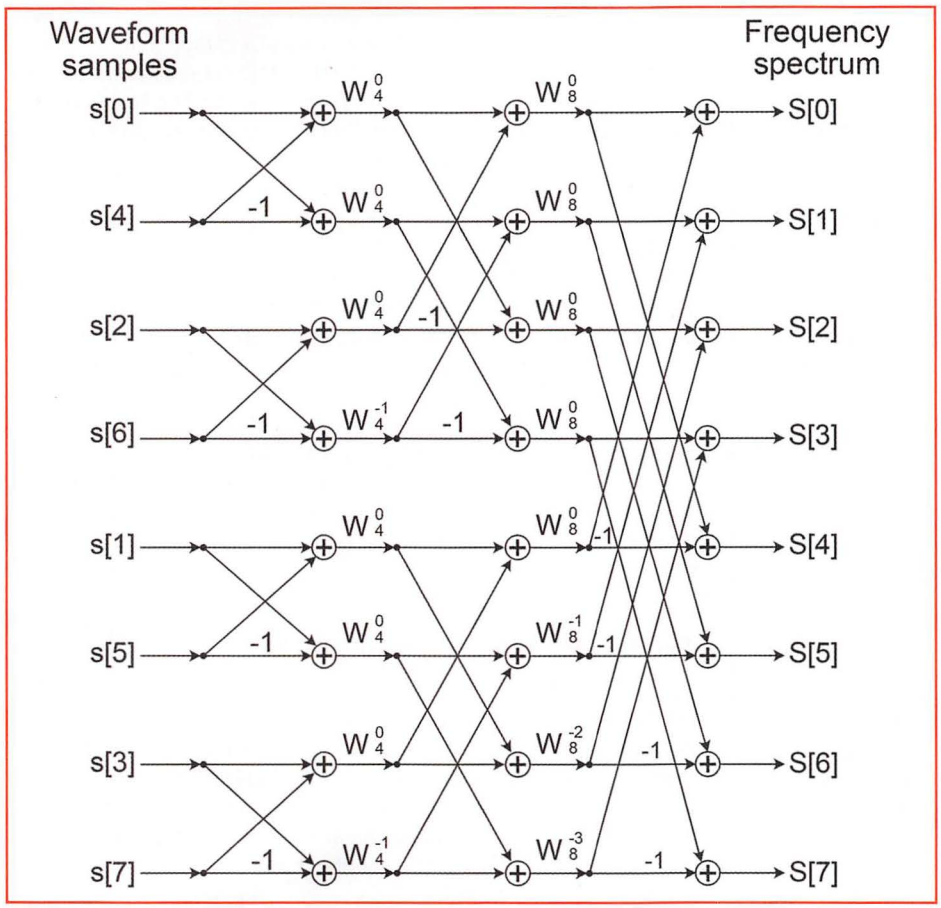

FIGURE 2: Diagrammatic view of an 8-point FFT, showing how the re-ordered (bit reversed) input data is reduced to multiple two-point DFTs in a tree structure.

Although precise details of the FFT are not described here, **Appendix 1** shows a listing of an FFT algorithm in one dialect of the BASIC language. The same buffer is used for input and output. Data enter and exit the routine in the array `dat()`, which contains alternate I and Q samples starting at an index of 1. For a simple serial-input stream of samples, every alternate location (I data) is filled with these, and the Q samples in-between are set to zero. The BASIC programme was derived from a FORTRAN process dating from the mid nineteen sixties, and the bit-reversal process carried out at the start may not be the most-efficient way of doing this. As bit reversal and reordering is just a small part of the processing overhead, no further optimisation was made in this part of the FFT routine.

Figure 2 shows a diagrammatic view of the process for an 8-point FFT, illustrating how samples are reordered, multiplied by the weightings (the sine and cosine values, labelled W_N) and summed in the circles with a plus sign. Note how the input sample numbers on the left have been re-ordered 0, 4, 2, 6, 1, 5, 3, 7. The binary equivalents of these are 000, 100, 010, 110, 001, 101, 011, 111, and it is clear that this is a bit-reversal of the original order 0, 1, 2, 3, 4, 5, 6, 7 or in binary 000, 001, 010, 011, 100, 101, 110, 111.

Windowing

The DFT or FFT process works on a fixed-length block of input samples, whilst the input data is of an arbitrary waveform shape. The start and finish values in the buffer will be different, whereas all the multiple sine and cosine reference frequencies assume a perfect start and finish. This causes a bit of 'leakage', or non-zero summations, where the result would otherwise sum to zero if start- and end-values had been the same. The effect in the frequency domain is to raise the noise floor and limit the dynamic range that can be detected. The solution is to multiply the set of input samples by a 'window' that reduces the start- and finish-samples to near zero, keeping those in the middle the same. The frequency resolution is then reduced, and two, or even three, bins can be affected by one frequency component. There is a compromise between dynamic range and effective frequency resolution (ie the number of bins affected), depending on the window type in use. Using no window, for example, offers the maximum frequency resolution of one bin, but limits the dynamic range potentially to 13dB. A Hamming window is usually taken as a good compromise, and allows around 40dB dynamic range with a resolution of about two bins. A more-extreme window is the Blackman-Harris window that offers 90dB dynamic range, but a frequency resolution of three bins. This is why most spectral-analysis and digital-receiver software allows the user to choose the window type, to optimise frequency spectral data or dynamic range.

Complex, or I/Q data

So far, we have mentioned sine and cosine as being required to cope with arbitrary phase shifts, but have assumed just a single stream of input samples. The buffer is N samples long but, because of the Nyquist theorem, we can only generate N/2 spectral components, up to half the sampling rate. We are using an in-situ calculation, where the same buffer memory contains input samples and the output spectrum. If it is N samples long, but we only get N/2 spectral components, what lies in the other N/2 memory locations? The answer is a mirror image of the spectrum, a repeat of all the results, but in the opposite direction. The FFT process calculates positive and negative frequencies and a single stream of input samples contains both of these – they are the same – so the spectral output has to show both.

The interesting situation comes if we perform the FFT on I/Q pairs of input samples. We know that I/Q data, depending on the relative phase of the pairs, whether they are 90° or -90° apart, indicates negative or positive frequency. Such an I/Q steam of samples is exactly what we get when an RF signal is down-converted with a quadrature local oscillator in a pair of mixers.

The FFT has 'knowledge' of all negative and positive frequency components present in the input samples, by virtue of their phases, and the resulting spectrum reflects this. Locations 0 to N/2 of the in-situ buffer, after the routine has finished, contain the spectrum of positive frequencies, exactly as it did before, but now the locations from N/2 to N-1 contain the negative frequencies. (Don't be distracted by wondering what a 'negative' frequency is. It is just a part of the complex algebra, and can be displayed on the negative axis of a graph of the spectrum. The frequency is the frequency!) To get a proper continuous spectral display, these upper locations need to be shifted backwards so the last one (location N-1) buts up against location zero. The complete spectrum now runs from $-F_s/2$ through zero to $+F_s/2$. By using quadrature inputs and the full FFT output, we have generated a bandwidth of spectral information equal to the sampling rate. But recall that to do this we must supply I/Q data to start with. So such full-bandwidth spectrum processing is only available for things like digital receivers where the 'negative frequency' is there in the first place, from a quadrature down-converter.

```
  SUB FFT(numsamples&)      'Data in format (I, Q, I, Q...) starting
at index 1'
    nn& = 2 * NumSamples&  'Needed because of using one linear
array for I and Q'
    j& = 1
    FOR i& = 1 TO nn& STEP 2      'Bit reversal first'
      IF j& > i& THEN
        SWAP dat(i&) , dat(j&)
        SWAP dat(i& + 1) , dat(j& + 1)
      END IF
      m& = nn& \ 2          'backslash     is     integer-divide
operation'
      WHILE m& >= 2 AND j& > m&
        j& = j& - m&
        m& = m& \ 2
      WEND
      j& = j& + m&
    NEXT i&
    mmax& = 2

    WHILE nn& > mmax&             'Now do the transform'
      istep& = 2 * mmax&
      theta = 2 * pi / mmax&
      wpr = COS(theta)      'cos  and  sin  value  could  be  pre-
calculated...'
      wpi = SIN(theta)      ' ... and stored in a table for speed.'
      wr = 1
      wi = 0

      FOR m& = 1 TO mmax& STEP 2
        FOR i& = m& TO nn& STEP istep&  'DFT butterflies'
          j& = i& + mmax&
          tempr = wr * dat(j&) - wi * dat(j& + 1)
          tempi = wr * dat(j& + 1) + wi * dat(j&)
          dat(j&) = dat(i&) - tempr
          dat(j& + 1) = dat(i& + 1) - tempi
          dat(i&) = dat(i&) + tempr
          dat(i& + 1) = dat(i& + 1) + tempi
        NEXT i&
        wtemp = wr
        wr = wr * wpr - wi * wpi
        wi = wi * wpr + wtemp * wpi
      NEXT m&
      mmax& = istep&
    WEND
  END SUB
```

FIGURE 3: A routine for in-situ radix-2 FFT.

Normal audio input, say from a microphone, can only be processed to a bandwidth of $F_s/2$.

Variations and other types of FFT

Here we have described the radix-2 FFT process, where a buffer length has to be a power of two. This may not always be ideal, and the FFT process can be performed with other radix values. The process is then more complicated to understand, with the reordering particularly so but, where results may only be wanted for a certain range of output frequency data, radix-3, radix-5 or even radix-7 FFTs are possible, or some product of these like radix-6. The buffer length then needs to be a product or power of the radix, which may have certain processing advantages. It is no coincidence that for all the modes within the WSJT-X suite of data modes, all the symbol rates and tone spacings, are exact submultiples of the sampling rate, and these submultiples all have factors of just 2, 3 and 5. FFTs with different radixes form a fundamental part of the decoding process in this software.

The examples given have adopted in-situ calculation where the same buffer is used for time and resulting frequency data. This may not always be convenient, so there is no reason why separate buffers for time-sampled data and calculated frequency bins can't be maintained.

Continuous conversion

The process described above, reading one complete buffer full of input samples, doing the transform, then moving to a completely new set of samples is suitable for the situation where the type of input waveform is static, such as continuous tones and constant-amplitude waveforms that have a constant frequency spectrum. But in reality we want to look at waveforms that change all the time, sometimes quite rapidly, like the RF spectrum, voice waveforms, or the like. If we use the one-buffer-at-a-time process, we are in danger of missing events that change suddenly, especially those that occur close to the boundaries of each block.

To allow maximum opportunity to capture rapid changes, it is usual to perform an FFT on a block, then read in a set of just half a block of new samples, moving the remaining half of the old ones to the end and discarding the other half. Now the FFT process must be performed twice as fast, on overlapping pairs of half-block lengths of data, so each half-buffer of time samples appears in two successive frequency spectra. Occasionally even further overlapping is employed with three, four or even more sub-blocks being employed, with a corresponding increase in the FFT rate.

When overlapping data is worked on in this way, simply reading it into a buffer for in-situ processing is not an option as a portion of that input data needs to be retained for the subsequent transform. The new samples have to be copied into the FFT working buffer from a separate buffer containing all the input samples. This is another application for the circular buffer technique introduced in the digital filtering article. The FFT buffer is then filled starting from the tail pointer of a suitably-long circular buffer.

Appendix 1

Figure 3 shows a routine for in-situ radix-2 FFT written in one dialect of BASIC derived from an original FORTRAN routine. Data is passed into and out of the routine in the global array `dat()` which contains alternate real and imaginary samples, or successive I/Q pairs. The index starts at 1 and continues to 2N for N complex samples. For real-only data, the input is read into alternate 'I' locations with the 'Q' values each set to zero. The bit reversal is carried out first, in a seemingly obscure way in this routine but it works fast enough. Variables ending in '&' are integer values, and those without a suffix are floating-point values.